Name

t_accept – accept a connect request

Syntax

#include <xti.h>

int t_accept(*fd, resfd, call***)**
int *fd*;
int *resfd*;
struct t_*call* ***call*;

Arguments

fd Identifies the local transport endpoint where the connect indication arrived.

resfd Specifies the local transport endpoint where the connection is to be established.

call Contains information required by the transport provider to complete the connection.

The *Call* argument points to a **t_call** structure that contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

In *call*, the members have the following meanings:

addr Specifies the address of the caller.

opt Indicates any protocol-specific parameters associated with the connection.

udata Points to any user data to be returned to the caller.

sequence Is the value returned by `t_listen()` that uniquely associates the response with a previously received connect indication.

Description

A transport user issues this function to accept a connect request. A transport user can accept a connection on either the same, or on a different local transport endpoint than the one on which the connect indication arrived. Before the connection can be accepted on the same endpoint (*resfd==fd*), the user must have responded to any previous connect indications received on that transport endpoint by means of `t_accept()` or `t_snddis()`. Otherwise, `t_accept()` fails and sets **t_errno** to [TBADF].

If a different transport endpoint is specified (*resfd!=fd*), the endpoint must be bound to a protocol address (if it is the same, *qlen* must be set to 0) and must be in the T_IDLE state before the `t_accept()` is issued.

For both types of endpoints, `t_accept()` fails and sets **t_errno** to [TLOOK] if there are connection indications, (for example, connect or disconnect) waiting to be received on that endpoint.

The values of parameters specified by *opt* and the syntax of those values are protocol-specific. The *udata* argument enables the called transport user to send user data to the caller and the amount of user data must not exceed the limits supported by the transport provider as returned in the *connect* field of the *info* argument of `t_open()` or `t_getinfo()`. If the *len* field of *udata* is zero, no data is sent to the caller.

All the *maxlen* fields are meaningless.

Parameters	Before Call	After Call
fd	x	/
resfd	x	/
call->addr.maxlen	/	/
call->addr.len	x	/
call->addr.buf	?(?)	/
call->opt.maxlen	/	/
call->opt.len	x	/
call->opt.buf	?(?)	/
call->udata.maxlen	/	/
call->udata.len	x	/
call->udata.buf	?(?)	/
call->sequence	x	/

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF] The file descriptor *fd* or *resfd* does not refer to a transport endpoint, or the user is illegally accepting a connection on the same transport endpoint on which the connect indication arrived.

[TOUTSTATE] The function was issued in the wrong sequence on the transport endpoint referenced by *fd*, or the transport endpoint referred to by *resfd* is not in the appropriate state.

[TACCES] The user does not have permission to accept a connection on the responding transport endpoint or to use the specified options.

[TBADOPT] The specified options were in an incorrect format or contained illegal information.

Name

intro – introduction to the X/Open Transport Interface (XTI)

Description

The X/Open Transport Interface defines a transport service interface that is independent of any specific transport provider. The interface is provided by way of a set of library functions for the C programming language.

Transport Providers

The transport layer can comprise one or more transport providers at the same time. The transport provider identifier parameter passed to the `t_open()` function determines the required transport provider.

Transport Endpoints

A transport endpoint specifies a communication path between a transport user and a specific transport provider, which is identified by a local file descriptor *(fd)*. When a user opens a transport provider identifier, a local file descriptor *fd* is returned that identifies the transport endpoint.

Synchronizing Endpoints

One process can simultaneously open several *fd*s. In synchronous mode, however the process must manage the different actions of the associated transport connections sequentially. Conversely, several processes can share the same *fd* (by `fork()` or `dup()` operations) but they have to synchronize themselves so as not to issue a function that is unsuitable to the current state of the transport endpoint.

Modes Of Service

The transport service interface supports two modes of service: connection mode and connectionless mode. A single transport endpoint cannot support both modes of service simultaneously.

The connection-mode transport service is circuit-oriented and enables data to be transferred over an established connection in a reliable, sequential manner. In contrast, the connectionless-mode transport service is message-oriented and supports data transfer in self-contained units with no logical relationship required among multiple units.

Error Handling

Two levels of error are defined for the transport interface. The first is the library error level. Each library function has one or more error returns. A return of -1 indicates a failure. An external integer, **t_errno**, which is defined in the header file **<xti.h>**, holds the specific error number when such a failure occurs. This value is set when errors occur but is not cleared on successful library calls, so it should be tested only after an error has been indicated. If implemented, a diagnostic function, **t_error**, prints out information on the current transport error. The state of the transport provider may change if a transport error occurs.

The second level of error is the operating system service routine level. A special library level error number has been defined called [TSYSERR], which is generated by each library function when the operating system service routine fails or some general error occurs. When a function sets **t_errno** to [TSYSERR], the specific system error can be accessed through the external variable **errno**.

Key For Parameter Arrays

Each XTI function description, includes an array that summarizes the content of the input and output parameter. The key is as follows:

Key	Description
x	The parameter value is meaningful (input parameter must be set before the call and output parameter must be read after the call).
(x)	The content of the object pointed by the x pointer is meaningful.
?	The parameter value is meaningful, but the parameter is oprtional.
(?)	The content of the object pointed by the ? pointer is optional.
/	The parameter value is meaningless.
=	After the call, the parameter keeps the same value as before the call.

[TBADDATA]	The specific amount of user data was not within the bounds allowed by the transport provider.
[TBADADDR]	The specified protocol address was in an incorrect format or contained illegal information.
[TBADSEQ]	The specified sequence number was invalid.
[TLOOK]	An asynchronous event has occurred on the transport endpoint referenced by *fd* and requires immediate attention.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

t_connect(3xti), t_getstate(3xti), t_listen(3xti), t_open(3xti) , t_optmgmt(3xti), t_rcvconnect(3xti)

Name

t_alloc – allocate a library structure

Syntax

```
#include <xti.h>

char *t_alloc(fd, struct_type, fields)
int fd;
int struct_type;
int fields;
```

Arguments

fd Refers to the transport endpoint through which the newly allocated structure is passed.

struct_type Specifies the allocated structure where each structure can subsequently be used as an argument to one or more transport functions.

The *struct_type* argument must specify one of the following:

T_BIND_STR	*struct*	t_bind
T_CALL_STR	*struct*	t_call
T_OPTMGMT_STR	*struct*	t_optmgmt
T_DIS_STR	*struct*	t_discon
T_UNITDATA_STR	*struct*	t_unitdata
T_UDERROR_STR	*struct*	t_uderr
T_INFO_STR	*struct*	t_info

fields Specifies which buffers to allocate, where the argument is the bitwise-OR of any of the following:

T_ADDR The *addr* field of the t_bind, t_call, t_unitdata, or **t_uderr** structures (size obtained from *info_addr*).

T_OPT The *opt* field of the **t_optmgmt**, **t_call**, **t_unitdata**, or **t_uderr** structures (size obtained from *info_options*).

T_UDATA The *udata* field of the **t_call**, **t_discon**, or **t_uderr** structures (for T_CALL_STR, size is the maximum value of *info_connect* and *info_discon*; for T_DIS_STR, size is the value of *info_discon*; for T_UNITDATA_STR, size is the value of *info_tsdu*).

T_ALL All relevant fields of the given structure.

Description

The t_alloc() function dynamically allocates memory for the various transport function argument structures as listed under the ARGUMENTS section. This function allocates memory for the specified structure and also allocates memory for buffers referenced by the structure.

Each of the accepted structures, except t_info(), contains at least one field of type *struct netbuf*. For each field of this type, the user can specify that the buffer for that field should be allocated as well. The length of the buffer allocated is based on the

size information returned in the `t_open()` or `t_getinfo()`.

For each field specified in *fields*, `t_alloc()` allocates memory for the buffer associated with the field and initializes the *len* field to zero and the *buf* pointer and *maxlen* field accordingly. Because the length of the buffer allocated is based on the same size information that is returned to the user on `t_open()` and `t_getinfo()`, *fd* must refer to the transport endpoint through which the newly allocated structure will be passed. In this way, the appropriate size information can be accessed. If the size value associated with any specified field is –1 or –2, `t_alloc()` will be unable to determine the size of the buffer to allocate and will fail, setting **t_errno** to [TSYSERR] and **errno** to [EINVAL]. For any field not specified in *fields*, *buf* will be set to NULL and *maxlen* will be set to zero.

Use of `t_alloc()` to allocate structures helps to ensure the compatibility of user programs with future releases of the transport interface functions.

Parameters	Before Call	After Call
fd	x	/
struct_type	x	/
fields	x	/

Return Value

Upon successful completion, `t_alloc()` returns a pointer to the newly allocated structure. On failure, NULL is returned.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TNOTSUPPORT]	This function is not supported by the current implementation of XTI.
[TSYSERR]	A system error has occurred during execution of this function.
[TNOSTRUCTYPE]	An unsupported *struct_type* has been requested.

See Also

t_free(3xti), t_getinfo(3xti), t_open(3xti)

Name

t_bind – bind an address to a transport endpoint

Syntax

```
#include <xti.h>

int t_bind(fd, req, ret)
int fd;
struct t_bind *req;
struct t_bind *ret;
```

Arguments

fd Refers to the transport endpoint which will be associated with a protocol address.

req Points to a **t_bind** structure containing the following members:

```
struct netbuf addr;
unsigned qlen;
```

The *addr* field of the `t_bind()` structure specifies a protocol address, and the *qlen* field is used to indicate the maximum number of outstanding connect indications.

ret Points to a `t_bind()` structure. See the *req* argument.

Description

This function associates a protocol address with the transport endpoint specified by *fd* and activates the transport endpoint. In connection mode, the transport provider can begin enqueuing incoming connect indications or servicing a connection request on the transport endpoint. In connectionless mode, the transport user can send or receive data units through the transport endpoint.

Parameters	Before Call	After Call
fd	x	/
req->addr.maxlen	/	/
req->addr.len	x>=0	/
req->addr.buf	x(x)	/
req->qlen	x>=0	/
ret->addr.maxlen	x	/
ret->addr.len	/	x
ret->addr.buf	x	(x)
ret->qlen	/	x>=0

The *req* argument is used to request that an address, represented by the **netbuf** structure, be bound to the given transport endpoint. The *len* specifies the number of bytes in the address, and *buf* points to the address buffer. The *maxlen* has no meaning for the *req* argument. On return, *ret* contains the address that the transport provider actually bound to the transport endpoint; this may be different from the address specified by the user in *req*. In *ret*, the user specifies *maxlen*, which is the maximum

size of the address buffer, and *buf*, which points to the buffer where the address is to be placed. On return, *len* specifies the number of bytes in the bound address, and *buf* points to the bound address. If *maxlen* is not large enough to hold the returned address, an error results.

If the requested address is not available, or if no address is specified in *req* (the *len* field of *addr* in *req* is zero), the transport provider assigns an appropriate address to be bound only if automatic generation of an address is supported and returns that address in the *addr* field of *ret*. The user can compare the addresses in *req* and *ret* to determine whether the transport provider bound the transport endpoint to a different address than that requested. In any XTI implementation, if the `t_bind()` function does not allocate a local transport address, then the returned address is always the same as the input address and the structure *req->addr* must be filled by the user before the call. If the local address is not furnished for the call (*req->addr.len=0*), the `t_bind()` returns –1 with **t_errno** set to [TNOADDR].

The *req* may be NULL if the user does not wish to specify an address to be bound. Here, the value of *qlen* is assumed to be zero, and the transport provider must assign an address to the transport endpoint. Similarly, *ret* may be NULL if the user does not care what address was bound by the provider and is not interested in the negotiated value of *qlen*. It is valid to set *req* and *ret* to NULL for the same call, in which case the provider chooses the address to bind to the transport endpoint and does not return the information to the user.

The *qlen* field has meaning only when initializing a connection-mode service. It specifies the number of outstanding connect indications the transport provider should support for the given transport endpoint. An outstanding connect indication is one that has been passed to the transport user by the transport provider but has not been accepted or rejected. A value of *qlen* greater than zero is meaningful only when issued by a passive transport user that expects other users to call it. The value of *qlen* will be negotiated by the transport provider and may be changed if the transport provider cannot support the specified number of outstanding connect indications. On return, the *qlen* field in *ret* contains the negotiated value.

This function allows more than one transport endpoint to be bound to the same protocol address. The transport provider, however, must support this capability also, it is not allowable to bind more than one protocol address to the same transport endpoint. If a user binds more than one transport endpoint to the same protocol address, only one endpoint can be used to listen for connect indications associated with the protocol address.

In other words, only one `t_bind()` for a given protocol address can specify a value of *qlen* greater than zero. In this way, the transport provider can identify which transport endpoint should be notified of an incoming connect indication. If a user attempts to bind a protocol address to a second transport endpoint with a value of *qlen* greater than zero, the transport provider assigns another address to be bound to that endpoint or, if automatic generation of addresses is not supported, returns –1 and sets **t_errno** to [TADDRBUSY].

When a user accepts a connection on the transport endpoint that is being used as the listening endpoint, the bound protocol address will be found to be busy for the duration of the connection, until a `t_unbind()` or `t_close()` call has been issued. No other transport endpoints may be bound for listening on that same protocol address while that initial listening endpoint is active (in the data transfer phase or in the T_IDLE state). This prevents more than one transport endpoint bound

to the same protocol address from accepting connect indications.

Return Value

Upon successful completion, `t_bind()` returns 0 and –1 on failure, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TOUTSTATE]	The function was issued in the wrong sequence.
[TBADADDR]	The specified protocol address was in an incorrect format or contained illegal information.
[TNOADDR]	The transport provider could not allocate an address.
[TACCES]	The user does not have permission to use the specified address.
[TBUFOVFLW]	The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The provider's state changes to T_IDLE and the information to be returned in *ret* is discarded.
[TSYSERR]	A system error has occurred during execution of this function.
[TADDRBUSY]	The address requested is in use and the transport provider cannot be allocate a new address.

See Also

t_alloc(3xti), t_close(3xti), t_open(3xti), t_optmgmt(3xti), t_unbind(3xti)

Name

t_close – close a transport endpoint

Syntax

#include <xti.h>

int t_close *fd*)
int *fd*;

Arguments

fd Identifies the local transport endpoint.

Description

The `t_close()` function informs the transport provider that the user is finished with the transport endpoint specified by *fd* and frees any local library resources associated with the endpoint. In addition, `t_close()` closes the file associated with the transport endpoint.

The `t_close()` function should be called from the T_UNBND state. However, this function does not check state information, so it can be called from any state to close a transport endpoint. If this occurs, the local library resources associated with the endpoint are freed automatically. In addition, `close()` is issued for that file descriptor; the `t_close()` abortives if there are no other descriptors in this or in another process that references the transport endpoint and breaks the transport connection that may be associated with that endpoint.

Parameters	Before Call	After Call
fd	x	/

Return Value

The `t_close` returns 0 on success and –1 on failure, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to the following:

The specified file descriptor does not refer to a transport endpoint.

See Also

t_getstate(3xti), t_open(3xti), t_unbind(3xti)

Name

t_connect – establish a connection with another transport user

Syntax

```
#include <xti.h>

int t_connect(fd, sndcall, rcvcall)
int fd;
struct t_call *sndcall;
struct t_call *rcvcall;
```

Arguments

fd	Identifies the local transport endpoint where communications is established.
sndcall	Specifies information needed by the transport provider to establish a connection.
rcvcall	Specifies information that is associated with the newly established connection.

The *sndcall* and *rcvcall* point to a **t_call** structure that contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

Description

This function enables a transport user to request a connection to the specified destination transport user. This function can be issued only in the T_IDLE state.

In *sndcall*, the argument *addr* specifies the protocol address of the destination transport user. The *opt* argument presents any protocol-specific information that might be needed by the transport provider. The *udata* argument points to optional user data that may be passed to the destination transport user during connection establishment. The *sequence* argument has no meaning for this function.

On return in *rcvcall*, *addr* argument returns the protocol address associated with the responding transport endpoint. The *opt* argument presents any protocol-specific information associated with the connection. The *udata* argument points to optional user data that may be returned by the destination transport user during connection establishment. The *sequence* argument has no meaning for this function.

The *opt* argument permits users to define the options that can be passed to the transport provider. These options are specific to the underlying protocol of the transport provider. The user can choose not to negotiate protocol options by setting the *len* field of *opt* to zero. In this case, the provider may use default options.

Parameters	Before Call	After Call
resfd	x	/
sndcall->addr.maxlen	/	/
sndcall->addr.len	x	/
sndcall->addr.buf	x(x)	/
sndcall->opt.maxlen	/	/
sndcall->opt.len	x	/
sndcall->opt.buf	?(?)	/
sndcall->udata.maxlen	/	/
sndcall->udata.len	x	/
sndcall->udata.buf	?(?)	/
sndcall->sequence	/	/
rcvcall->addr.maxlen	x	/
rcvcall->addr.len	/	x
rcvcall->addr.buf	x	(x)
rcvcall->opt.maxlen	x	/
rcvcall->opt.len	/	x
rcvcall->opt.buf	x	(x)
rcvcall->udata.maxlen	x	/
rcvcall->udata.len	/	x
rcvcall->udata.buf	x	(?)
rcvcall->sequence	/	/

If used, **sndcall->opt.buf** structure must point to the corresponding options structures (**isoco_options**, **isocl_options** or **tcp_options**). The *maxlen* and *buf* fields of the **netbuf** structure pointed by *rcvcalladdr* and *rcvcall->opt* must be set before the call.

The *udata* argument enables the caller to pass user data to the destination transport and receive user data from the destination user during connection establishment. However, the amount of user data must not exceed the limits supported by the transport provider as returned in the *connect* field of the *info* argument of `t_open()`. If the *len* of *udata* is zero in *sndcall*, no data are sent to the destination transport user.

On return, the *addr*, *opt*, and *udata* fields of *rcvcall* updates to reflect values associated with the connection. Thus, the *maxlen* field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, *rcvcall* can be NULL, in which case no information is given to the user on return from `t_connect()`.

By default, `t_connect()` executes in synchronous mode and waits for the destination user's response before returning control to the local user. A successful return (that is, a return value of zero) indicates that the requested connection has been established. However, if O_NONBLOCK is set by means of `t_open()` or `fcntl()`, `t_connect()` executes in asynchronous mode. In this case, the call waits for the remote user's response but returns control immediately to the local user and returns –1 with **t_errno** set to [TNODATA] to indicate that the connection has

not yet been established. In this way, the function simply initiates the connection establishment procedure by sending a connect request to the destination transport user. The `t_rcvconnect()` function is used in conjunction with `t_connect()` to determine the status of the requested connection.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TOUTSTATE]	The function was issued in the wrong sequence.
[TNODATA]	O_NONBLOCK was set, so the function successfully initiated the connection establishment procedure but did not wait for a response from the remote user.
[TACCES]	The user does not have permission to use the specified address or options.
[TBADOPT]	The specified protocol options were in an incorrect format or contained illegal information.
[TBADADDR]	The specified protocol address was in an incorrect format or contained illegal information.
[TBADDATA]	The amount of user data specified was not within the bounds allowed by the transport provider.
[TBUFOVFLW]	The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. If executed in synchronous mode, the provider's state, as seen by the user, changes to T_DATAXFER, and the connect indication information to be returned in *rcvcall* is discarded.
[TLOOK]	An asynchronous event has occurred on this transport endpoint and requires immediate attention.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

t_accept(3xti), t_alloc(3xti), t_getinfo(3xti), t_listen(3xti), t_open(3xti), t_optmgmt(3xti), t_rcvconnect(3xti)

Name

t_error – produces error message

Syntax

#include <xti.h>

int t_error(*errmsg*)
char **errmsg*;
extern char **t_errlist[]*;
extern int *t_nerr*;

Arguments

errmsg Is a user-supplied error message that gives context to the error.

Description

The `t_error()` function produces a message on the standard error output that describes the last error encountered during a call to a transport function.

The `t_error()` function prints the user-supplied error message followed by a colon and a standard error message for the current error defined in **t_errno**. If **t_errno** is [TSYSERR], `t_error()` also prints a standard message for the current value contained in **errno**.

To simplify variant formatting of messages, the array of message strings **t_errlist** is provided: **t_errno** can be used as an index in this table to get the message string without the newline. The *t_nerr* is the largest message number provided for in the *t_errlist* table.

The **t_errno** variable is set only when an error occurs and is not cleared on successful calls.

Parameters	Before Call	After Call
errmsg	x	/

Examples

If a `t_connect()` function fails on transport endpoint *fd2* because a bad address was given, the following call may follow the failure:

```
t_error ("t_connect failed on fd");
```

The diagnostic message to be printed would look like:

```
t_connect failed on fd2: Incorrect transport address format
```

where "Incorrect transport address format" identifies the specific error that occurred, and "t_connect failed on fd2" tells the user which function failed on which transport endpoint.

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to the following:

[TNOTSUPPORT] This function is not supported by the current implementation of XTI.

Name

t_free – free a library structure

Syntax

#include <xti.h>

int t_free(*ptr, struct_type*)
char **ptr*;
int *struct_type*;

Arguments

ptr Points to one of the seven structure types described for `t_alloc()`.

struct_type Identifies the type of that structure, which must be one of the following:

`T_BIND_STR`	*struct*	**t_bind;**
`T_CALL_STR`	*struct*	**t_call**
`T_OPTMGMT_STR`	*struct*	**t_optmgmt**
`T_DIS_STR`	*struct*	**t_discon**
`T_UNITDATA_STR`	*struct*	**t_unitdata**
`T_UDERROR_STR`	*struct*	**t_uderr**
`T_INFO_STR`	*struct*	**t_info**

Each of these structures is used as an argument to one or more transport functions.

Description

The `t_free()` function frees memory previously allocated by `t_alloc()`. This function frees memory for the specified structure and also frees memory for buffers referenced by the structure.

Parameters	Before Call	After Call
ptr	x	/
struct_type	x	/

The `t_free()` function checks the *addr*, *opt*, and *udata* fields of the given structure (as appropriate) and free the buffers pointed to by the *buf* field of the **netbuf** structure. If *buf* is NULL, `t_free()` does not attempt to free memory. After all buffers are freed, `t_free()` frees the memory associated with the structure pointed to by *ptr*.

Results are undefined if *ptr* or any of the *buf* pointers points to a block of memory not previously allocated by `t_alloc()`.

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TNOTSUPPORT]	This function is not supported by the current implementation of XTI.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

t_alloc(3xti)

Name

t_getinfo – get protocol-specific service information

Syntax

```
#include <xti.h>

int t_getinfo(fd, info)
int fd;
struct t_info *info;
```

Arguments

fd Identifies the file descriptor that is associated with the underlying transport protocol from which the current characteristics are to be returned.

info Specifies the structure that is used to return the same information returned by `t_open()`. Points to a **t_info** structure which contains the following members:

```
long addr;      /* max size of the transport protocol address */
long options;   /* max number of bytes of protocol-specific options */
long tsdu;      /* max size of a transport service data unit (TSDU) */
long etsdu;     /* max size of an expedited transport service data unit
                   (ETSDU) */
long connect;   /* max amount of data allowed on connection
                   establishment functions */
long discon;    /* max amount of data allowed on t_snddis() and
                   t_rcvdis() functions */
long servtype;  /* service type supported by the transport provider */
```

The values of the fields have the following meanings:

addr A value greater than or equal to zero indicates the maximum size of a transport protocol address; a value of –1 specifies that there is no limit on the address size; and a value of –2 specifies that the transport provider does not provide user access to transport protocol addresses.

options A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the provider; a value of –1 specifies that there is no limit on the option size and a value of –2 specifies that the transport provider does not support user-settable options.

tsdu A value greater than zero specifies the maximum size of a transport service data unit (TSDU); a value of zero specifies that the transport provider does not support the concept of TSDU, although it does support the sending of

a data stream with no logical boundaries preserved across a connection; a value of –1 specifies that there is no limit on the size of a TSDU and a value of –2 specifies that the transfer of normal data is not supported by the transport provider.

etsdu A value greater than zero specifies the maximum size of an expedited transport service data unit (ETSDU); a value of zero specifies that the transport provider does not support the concept of ETSDU, although it it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of –1 specifies that there is no limit on the size of ETSDU; and a value of –2 specifies that the transfer of expedited data is not supported by the transport provider.

connect A value greater than or equal to zero specifies the maximum amount of data that may be associated with connection establishment functions; a value of –1 specifies that there is no limit on the amount of data sent during connection establishment; and a value of –2 specifies that the transport provider does not allow data to be sent with connection establishment functions.

discon A value greater than or equal to zero specifies the maximum amount of data that may be associated with the `t_snddis()` and `t_rcvdis()` functions; a value –1 specifies that there is no limit on the amount of data sent with these abortive release functions; and a value of –2 specifies that the transport provider does not allow data to be sent with the abortive release functions.

servtype This field specifies the service type supported by the transport provider, as described.

If a transport user is concerned with protocol independence, the sizes may be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the `t_alloc()` function can be used to allocate these buffers. An error results if a transport user exceeds the allowed data size on any function. The value of each field may change as a result of option negotiation, and `t_getinfo()` enables a user to retrieve the current characteristics of the underlying transport protocol.

The *servtype* field of *info* specifies one of the following values on return:

T_COTS The transport provider supports a connection-mode service but does not support the optional orderly release facility.

T_COTS_ORD
The transport provider supports a connection-mode service with the optional orderly release facility.

T_CLTS — The transport provider supports a connectionless-mode service. For this service type, `t_open()` returns –2 for ETSDU, `connect` and `discon`.

Description

This function returns the current characteristics of the underlying transport protocol associated with file descriptor *fd*. The *info* structure is used to return the same information returned by `t_open()`. This function enables a transport user to access this information during any phase of communications.

Parameters	Before Call	After Call
fd	x	/
info->addr	/	x
info->options	/	x
info->tsdu	/	x
info->etsdu	/	x
info->connect	/	x
info->discon	/	x
info->sertype	/	x

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF] — The specified file descriptor does not refer to a transport endpoint.

[TNOTSUPPORT] — This function is not supported by the current implementation of XTI.

[TSYSERR] — A system error has occurred during execution of this function.

See Also

t_alloc(3xti), t_open(3xti)

t_getstate (3xti)

Name

t_getstate – get the current state

Syntax

#include <xti.h>

int t_getstate(*fd***)**
int *fd***;**

Arguments

fd Identifies the local transport endpoint the current state is returned from.

Description

The `t_getstate()` function returns the current state of the transport provider associated with the transport endpoint specified by *fd*.

Parameters	Before Call	After Call
fd	x	/

Return Value

Upon successful completion, `t_getstate()` returns the current state. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error. The current state is one of the following:

T_UNBND Unbound

T_IDLE Idle

T_OUTCON Outgoing connection pending

T_INCON Incoming connection pending

T_DATAXFER Data transfer

T_OUTREL Outgoing orderly release (waiting for an orderly release indication)

T_INREL Incoming orderly release (waiting to send an orderly release request)

If the provider is undergoing a state transition when `t_getstate()` is called, the function fails.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint. This error may be returned when the *fd* has been previously closed or an erroneous number has been passed to the call.

[TSTATECHNG] The transport provider is undergoing a transient state change.

[TNOTSUPPORT]	This function is not supported by the current implementation of XTI.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

t_open(3xti)

Name

t_listen – listen for a connect request

Syntax

#include <xti.h>

int t_listen(*fd, call*)
int *fd*;
struct t_call **call*;

Arguments

fd Identifies the local transport endpoint where the connect indication arrived.

call Contains information describing the connect indication. The *call* points to a **t_call** structure which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

The members of the **t_call** structure have the following meanings:

addr Returns the protocol address of the calling transport user.

udata Returns any user data sent by the caller on the connect request.

sequence Identifies the returned connect indication with a unique number. The value of *sequence* enables the user to listen for multiple connect indications before responding to any of them.

Because this function returns values for the *addr*, *opt*, and *udata* fields of *call*, the *maxlen* field of each must be set before issuing the `t_listen()` to indicate the maximum size of the buffer for each.

Description

This function listens for a connect request from a calling transport user. The *fd* identifies the local transport endpoint where connect indications arrive. On return, *call* contains information describing the connect indication.

By default, `t_listen` executes in synchronous mode and waits for a connect indiction to arrive before returning to the user. However, if O_NONBLOCK is set by means of `t_open()` or `fcntl()`, `t_listen()` executes asynchronously, reducing to a poll for existing connect indications. If none are available, it returns –1 and sets `t_errno()` to [TNODATA].

Parameters	Before Call	After Call
fd	x	/
call->addr.maxlen	x	/
call->addr.len	/	x
call->addr.buf	x	(x)
call->opt.maxlen	x	/
call->opt.len	/	x
call->opt.buf	x	(x)
call->udata.maxlen	x	/
call->udata.len	/	x
call->udata.buf	x	(?)
call->sequence	/	x

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TOUTSTATE] The function was issued in the wrong sequence on the transport endpoint referenced by *fd*.

[TBADQLEN] The *qlen* of the endpoint referenced by *fd* is zero.

[TBUFOVFLW] The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. The provider's state, as seen by the user, changes to T_INCON, and the connect indication information to be returned in *call* is discarded. The value of *sequence* returned can be used to do a `t_snddis()`.

[TNODATA] O_NONBLOCK was set, but no connect indications had been queued.

[TLOOK] An asynchronous event has occurred on the transport endpoint and requires immediate attention.

[TNOTSUPPORT] This function is not supported by the underlying transport provider.

[TSYSERR] A system error has occurred during execution of this function.

See Also

fcntl(2), t_accept(3xti), t_alloc(3xti), t_bind(3xti), t_connect(3xti), t_open(3xti), t_optmgmt(3xti), t_rcvconnect(3xti)

Name

t_look – look at the current event on a transport endpoint

Syntax

```
#include <xti.h>

int t_look(fd)
int fd;
```

Arguments

fd	Identifies the transport endpoint where the current event is returned.

Description

This function returns the current event on the transport endpoint specified by *fd*. This function enables a transport provider to notify a transport user of an asynchronous event when the user is issuing functions in synchronous mode. Certain events require immediate notification of the user and are indicated by a specific error, [TLOOK], on the current or next function to be executed.

This function also enables a transport user to poll a transport endpoint periodically for asynchronous events.

Parameters	Before Call	After Call
fd	x	/

Return Value

Upon successful completion, `t_look()` returns a value that indicates which of the allowable events has occurred or returns zero if no event exists. One of the following events is returned:

T_LISTEN	Connection indication received
T_CONNECT	Connect confirmation received
T_DATA	Normal data received
T_EXDATA	Expedited data received
T_DISCONNECT	Disconnect received
T_UDERR	Datagram error indication
T_ORDREL	Orderly release indication
T_GODATA	Flow control restrictions on normal data flow have been lifted. Normal data can be sent again.
T_GOEXDATA	Flow control restrictions on expedited data flow have been lifted. Expedited data can be sent again.

On failure, –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

t_open(3xti), t_snd(3xti), t_sndudata(3xti)

Name

t_open – establish a transport endpoint

Syntax

```
#include <xti.h>

#include <fcntl.h>
int t_open(name, oflag, info)
char *name;
int oflag;
struct t_info *info;
```

Arguments

name Points to a transport provider identifier.

oflag Identifies any open flags as in `open()`. The *oflag* argument is constructed from O_RDWR optionally ORed with O_NONBLOCK. These flags are defined by the header file **<fcntl.h>**.

info Returns various default characteristics of the underlying transport protocol by setting fields in the *info* structure. This argument points to a `t_info()` structure that contains the following members:

```
long addr       /* max size of the transport protocol  address */
long options    /* max number of bytes of protocol specific options */
long tsdu       /* max size of a transport service data  unit (TSDU) */
long etsdu      /* max size of expedited transport  service data unit
                   (ETSDU) */
long connect    /* max amount of data allowed on connection
                   establishment functions */
long discon     /* max amount of data allowed on t_snddis() and
                   t_rcvdis() functions */
long servtype   /* service type supported by the transport provider */
```

The values of the fields have the following meanings:

addr A value greater than or equal to zero indicates the maximum size of a transport protocol address; a value of –1 specifies that there is no limit on the address size; and a value of –2 specifies that the transport provider does not provide user access to transport protocol addresses.

options A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the provider; a value of -1 specifies that there is no limit on the option size; and a value of -2 specifies that the transport provider does not support user-settable options.

tsdu — A value greater than zero specifies the maximum size of a transport service data unit (TSDU); a value of zero specifies that the transport provider does not support the concept of TSDU; although it does support the sending of a data stream with no logical boundaries preserved across a connection; a value of –1 specifies that there is no limit on the size of an ETSDU; and a value of –2 specifies that the transfer of normal data is not supported by the transport provider.

etsdu — A value greater than zero specifies the maximum size of an expedited transport service data unit (ETSDU); a value zero specifies that the transport provider does not support the concept of ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of –1 specifies that there is no limit on the size of an ETSDU; and a value –2 specifies that the transfer of expedited data is not supported by the transport provider.

connect — A value greater than or equal to zero specifies the maximum amount of data that may be associated with connection establishment functions; a value of –1 specifies that there is no limit on the amount of data sent during connection establishment; and a value of –2 specifies that the transport provider does not allow data to be sent with connection establishment functions.

discon — A value greater than or equal to zero specifies the maximum amount of data that may be associated with the `t_snddis()` and `t_rcvdis()` functions; a value of –1 specifies that there is no limit on the amount of data sent with these abortive release functions; and a –2 specifies that the transport provider does not allow data to be sent with abortive release functions.

servtype — This field specifies the service type supported by the transport provider, as described.

If a transport user is concerned with protocol independence, the sizes can be accessed to determine how large the buffers must be to hold each piece of information. Alternately, the `t_alloc()` function may be used to allocate these buffers. An error will result if a transport user exceeds the allowed data size on any function.

The *servtype* field of *info* specifies one of the following values on return.

T_COTS — The transport provider supports a connection-mode service but does not support the optional orderly release facility.

T_COTS_ORD — The transport provider supports a connection-mode service with the optional orderly release facility.

T_CLTS	The transport provider supports a connectionless-mode service. For this service type, `t_open()` returns –2 for *etsdu*, *connect*, and *discon*.

A single transport endpoint may support only one of the above services at one time. If *info* is set to NULL by the transport user, no protocol information is returned by `t_open()`.

Description

The `t_open()` function must be called as the first step in the initialization of a transport endpoint. This function establishes a transport endpoint by supplying a transport provider identifier that indicates a particular transport provider, that is a transport protocol, and returns a file descriptor that identifies that endpoint.

The `t_open()` function returns a file descriptor that is used by all subsequent functions to identify that particular local transport endpoint.

Parameters	Before Call	After Call
name	x	/
oflag	x	/
info->addr	/	x
info->options	/	x
info->tsdu	/	x
info->etsdu	/	x
info->connect	/	x
info->discon	/	x
info->servtype	/	x

Return Value

Upon successful completion, `t_open()` returns a file descriptor. On failure, -1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADFLAG]	An invalid flag is specified.
[TBADNAME]	Invalid transport provider name.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

open(2)

Name

t_optmgmt – manage options for a transport endpoint

Syntax

```
#include <xti.h>

int t_optmgmt(fd, req, ret)
int fd;
struct t_optmgmt *req;
struct t_optmgmt *ret;
```

Arguments

fd Identifies a bound transport endpoint.

req Points to a **t_optmgmt** structure. See also *ret* argument.

ret Points to a **t_optmgmt** structure containing the following members:

```
struct netbuf opt;
long flags;
```

The meanings of the fields are as follows:

opt Identifies protocol options.

flags
Specifies the action to take with these options.

The options are represented by a **netbuf** structure in a manner similar to the address in `t_bind()`. The *req* argument is used to request a specific action of the provider and to send options to the provider. The *len* field specifies the number of bytes in the options. The *buf* field points to the options buffer, and the *maxlen* field has no meaning for the *req* argument. The transport provider can return options and flag values to the user through *ret*. For *ret*, *maxlen* specifies the maximum size of the options buffer, and *buf* points to the buffer where the options are to be placed. On return, *len* specifies the number of bytes of options returned. The *maxlen* field has no meaning for the *req* argument, but must be set in the *ret* argument to specify the maximum number of bytes the option buffer can hold. The actual structure and content of the options is imposed by the transport provider.

The *flags* field of *req* must specify one of the following actions:

T_NEGOTIATE This action enables the user to negotiate the values of the options specified in *req* with the transport provider. The transport provider evaluates the requested options and negotiates the values, returns the negotiated values through *ret*.

T_CHECK This action enables the user to verify whether the options specified in *req* are supported by the transport provider. On return, the *flags* field of *ret* has either T_SUCCESS or T_FAILURE set to indicate to the user whether options are supported. These *flags* are only meaningful for the T_CHECK request.

T_DEFAULT This action enables a user to retrieve the default options supported by the transport provider into the *opt* field of *ret*. In *req*, the *len* field of *opt* must be zero and the *buf* field may be NULL.

Description

The `t_optmgmt()` function enables a transport user to receive, verify, or negotiate protocol options with the transport provider.

If issued as part of the connectionless-mode service, `t_optmgmt()` may block due to flow control constraints. That is, the function does not complete until the transport provider has processed all previously sent data units.

Parameters	Before Call	After Call
fd	x	/
req->opt.maxlen	/	/
req->opt.len	x	/
req->opt.buf	x(x)	/
req->flags	x	/
ret->opt.maxlen	x	/
ret->opt.len	/	x
ret->opt.buf	x	(x)
ret->flags	/	x

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TOUTSTATE] The function was issued in the wrong sequence.

[TACCES] The user does not have permission to negotiate the specified options.

[TBADOPT] The specified protocol options were in an incorrect format or contained illegal information.

[TBADFLAG] An invalid flag was specified.

[TBUFOVFLW] The number of bytes allowed for an incoming argument is not sufficient to store the value of that argument. The information to be returned in *ret* is discarded.

[TNOTSUPPORT] This function is not supported by the current implementation of XTI.

[TSYSERR] A system error has occurred during execution of this function.

See Also

t_accept(3xti), t_alloc(3xti), t_connect(3xti), t_getinfo(3xti), t_listen(3xti), t_open(3xti), t_rcvconnect(3xti)

Name

t_rcv – receive data or expedited data sent over a connection

Syntax

#include <xti.h>

int t_rcv(*fd, buf, nbytes, flags*)
int *fd*;
char **buf*;
unsigned *nbytes*;
int **flags*;

Arguments

fd	Identifies the local transport endpoint through which data arrives.
buf	Points to a receive buffer where user data is placed.
nbytes	Specifies the size of the receive buffer.
flags	Specifies optional flags. Can be set on return from `t_rcv()`.

Description

This function receives either normal or expedited data.

By default, `t_rcv()` operates in synchronous mode and waits for data to arrive if none is currently available. However, if O_NONBLOCK is set (by means of `t_open()` or `fcntl()`, `t_rcv()` executes in asynchronous mode and fails if no data is available.

On return from the call, if T_MORE is set in *flags* this indicates that there is more data and the current transport service data unit (TSDU) or expedited transport service data (ETSDU) must be received in multiple `t_rcv()` calls. Each `t_rcv()` with the T_MORE flag set indicates that another `t_rcv()` must follow immediately to get more data from the current TSDU. The end of the TSDU is identified by the return of a `t_rcv()` call with the T_MORE flag not set. If the transport provider does not support the concept of a TSDU as indicated in the *info* argument on return from `t_open()` or `t_getinfo()`, the T_MORE flag is not meaningful and should be ignored.

On return, the data returned is expedited data if T_EXPEDITED is set in *flags*. If the number of bytes of expedited data exceeds *nbytes*, `t_rcv()` sets T_EXPEDITED and T_MORE on return from the initial call. Subsequent calls to retrieve the remaining ETSDU have T_EXPEDITED set on return. The end of the ETSDU is identified by the return of a `t_rcv` call with the T_MORE flag not set.

If expedited data arrives after part of a TSDU has been retrieved, receipt of the remainder of the TSDU is suspended until the ETSDU has been processed. Only after the full ETSDU has been retrieved (T_MORE not set) will the remainder of the TSDU be available to the user.

In synchronous mode, the only way for the user to be notified of the arrival of normal or expedited data is to issue this function or check for the T_DATA or T_EXDATA events using the `t_look()` function.

Parameters	Before Call	After Call
fd	x	/
buf	x	(x)
nbytes	x	/
flags	/	x

Return Value

Upon successful completion, `t_rcv()` returns the number of bytes received. On failure, a value of -1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TOUTSTATE]	The function was issued in the wrong sequence on the transport endpoint referenced by *fd*.
[TNODATA]	O_NONBLOCK was set, but no data is currently available from the transport provider.
[TLOOK]	An asynchronous event has occurred on the transport endpoint and requires immediate attention.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

fcntl(2), t_getinfo(3xti), t_look(3xti), t_open(3xti), t_snd(3xti)

t_rcvconnect (3xti)

Name

t_rcvconnect – receive the confirmation from a connect request

Syntax

```
#include <xti.h>

int t_rcvconnect(fd, call)
int fd;
struct t_call *call;
```

Arguments

fd Identifies the local transport endpoint where communications is established.

call Contains information associated with the newly established connection. *Call* points to a t_*call* structure that contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

The members of the *t_call* structure have the following meanings:

addr Returns the protocol address associated with the responding transport endpoint.

opt Presents any protocol-specific information associated with the transport endpoint.

udata Points to any optional user data that may be returned by the destination transport user during connection establishment.

sequence Has no meaning for this function.

Description

This function enables a calling transport user to determine the status of a previously sent connect request. Is used in conjunction with `t_connect()` to establish a connection in asynchronous mode. The connection is established on successful completion of this function.

The *maxlen* field of each argument must be set before issuing this function to indicate the maximum size of the buffer for each. However, *call* can be NULL, in which case no information is given to the user on return from `t_rcvconnect()`. By default, `t_rcvconnect()` executes in synchronous mode and waits for the connection to be established before returning. On return, the *addr*, *opt*, and *udata* fields reflect values associated with the connection.

Parameters	Before Call	After Call
fd	x	/
call->addr.maxlen	x	/
call->addr.len	/	x
call->addr.buf	x	(x)
call->opt.maxlen	x	/
call->opt.len	/	x
call->opt.buf	x	(x)
call->udata.maxlen	x	/
call->udata.len	/	x
call->udata.buf	x	(?)
call->sequence	/	/

If O_NONBLOCK is set by means of `t_open()` or `fcntl()`, `t_rcvconnect()` executes in asynchronous mode and reduces to a poll for existing connect confirmations. If none is available, `t_rcvconnect()` fails and returns immediately without waiting for the connection to be established. The `t_rcvconnect()` function must be reissued at a later time to complete the connection establishment phase and retrieve the information returned to *call*.

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of -1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, `t_errno()` is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TBUFOVFLW]	The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument. The connect information to be returned in *call* is discarded. The provider's state, as seen by the user, is changed to DATAXFER.
[TNODATA]	O_NONBLOCK was set, but a connect confirmation has not yet arrived.
[TLOOK]	An asynchronous event has occurred on the transport connection and requires immediate attention.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TOUTSTATE]	The function was issued in the wrong sequence on the transport endpoint referenced by *fd*.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

t_accept(3xti), t_alloc(3xti), t_bind(3xti), t_connect(3xti), t_listen(3xti), t_open(3xti), t_optmgmt(3xti)

Name

t_rcvdis – retrieve information from disconnect

Syntax

#include <xti.h>

int t_rcvdis(*fd, discon***)**
int *fd*;
struct t_discon **discon*;

Arguments

fd Identifies the local transport endpoint.

discon Points to a **t_discon** structure containing the following members:

```
struct netbuf udata;
int reason;
int sequence:
```

The members of the *t_discon* struct have the following meanings:

udata Identifies any user data that was sent with the disconnect.

reason Specifies the reason for the disconnect through a protocol-dependent reason code.

sequence Identifies an outstanding connect indication with which the connection is associated. The *sequence* field is only meaningful when `t_rcvdis()` is issued by a passive transport user who has executed one or more t_listen() functions and is processing the resulting connect indications. If a disconnect indication occurs, *sequence* can be used to identify which of the outstanding connect indications is associated with the disconnect.

Description

This function is used to identify the cause of a disconnect and to retrieve any user data sent with the disconnect.

If a user does not care if there is incoming data and does not need to know the value of *reason* or *sequence*, *discon* may be NULL and any user data associated with the disconnect is discarded. However, if a user has retrieved more than one outstanding connect indication, by means of `t_listen()` and *discon* is NULL, the user will be unable to identify with which connect indication the disconnect is associated.

Parameters	Before Call	After Call
fd	x	/
discon->udata.maxlen	x	/
discon->udata.len	/	x
discon->udata.buf	x	(?)
discon->reason	/	x
discon->sequence	/	?

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of -1 is returned, and `t_errno()` is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF] The specified file descriptor *fd* does not refer to a transport endpoint.

[TOUTSTATE] The function was issued in the wrong sequence on the transport endpoint referenced by *fd*.

[TNODIS] No disconnect indication currently exists on the specified transport endpoint.

[TBUFOVFLW] The number of bytes allocated for incoming data is not sufficient to store the data. If *fd* is a passive endpoint with *ocnt* > 1, it remains in state T_INCON; otherwise, the endpoint state is set to T_IDLE. The disconnect indication information to be returned in *discon* will be discarded.

[TNOTSUPPORT] This function is not supported by the underlying transport provider.

[TSYSERR] A system error has occurred during execution of this function.

See Also

t_alloc(3xti), t_connect(3xti), t_listen(3xti), t_open(3xti), t_snddis(3xti)

Name

t_rcvrel – acknowledge receipt of an orderly release indication

Syntax

#include <xti.h>

int t_rcvrel(*fd*)
int *fd*;

Arguments

fd Identifies the local transport endpoint.

Description

This function is used to acknowledge receipt of an orderly release indication. After receipt of this indication, the user cannot attempt to receive more data, because such an attempt will block forever. However, the user can continue to send data over the connection if `t_sndrel()` has not been issued by the user.

This function is an optional service of the transport provider, and is only supported if the transport provider returned service type T_COTS_ORD on `t_open()` or `t_getinfo()`.

Parameters	Before Call	After Call
fd	x	/

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned, and **t_errno()** is set to indicate the error.

Diagnostics

On failure, `t_errno()` is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TOUTSTATE]	The function was issued in the wrong sequence on the transport endpoint referenced by *fd*.
[TNOREL]	No orderly release indication currently exists on the specified transport endpoint.
[TLOOK]	An asynchronous event has occurred on the transport endpoint and requires immediate attention.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

t_getinfo(3xti), t_open(3xti), t_sndrel(3xti)

Name

t_rcvudata – receive a data unit

Syntax

#include <xti.h>

int t_rcvudata(*fd, unitdata, flags*)
int *fd*;
struct t_unitdata **unitdata*;
int **flags*:

Arguments

fd Identifies the local transport endpoint through which data is received.

unitdata Holds information associated with the received data unit. The *unitdata* argument points to a **t_unitdata** structure containing the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata
```

On return from this call, the members have the following meanings:

addr Specifies the protocol address of the sending unit.

opt Identifies protocol-specific options that were associated with this data unit.

udata Specifies the user data that was received.

flags Set on return to indicate that the complete data unit was not received.

Description

This function is used in connectionless mode to receive a data unit from another transport user.

By default, `t_rcvudata()` operates in synchronous mode waits for a data unit to arrive if none is currently available. However, if O_NONBLOCK is set by means of `t_open()` or `fcntl()`, *udata* executes in asynchronous mode and fails if no data units are available.

The *maxlen* field of *addr*, *opt*, and *udata* must be set before issuing this function to indicate the maximum size of the buffer for each.

If the buffer defined in the *udata* field of *unitdata* is not large enough to hold the current data unit, the buffer fills and T_MORE sets in *flags* on return to indicate that another `t_rcvudata()` should be issued to retrieve the rest of the data unit. Subsequent `t_rcvudata()` calls return zero for the length of the address and options until the full data unit has been received.

Parameters	Before Call	After Call
fd	x	/
unitdata->addr.maxlen	x	/
unitdata->addr.len	/	x
unitdata->addr.buf	x	(x)
unitdata->opt.maxlen	x	/
unitdata->opt.len	/	x
unitdata->opt.buf	x	(x)
unitdata->udata.maxlen	x	/
unitdata->udata.len	/	x
unitdata->udata.buf	x	(x)
flags	/	x

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TOUTSTATE]	The function was issued in the wrong sequence on the transport endpoint referenced by *fd*.
[TNODATA]	O_NONBLOCK was set, but no data units are currently available from the transport provider.
[TBUFOVFLW]	The number of bytes allocated for the incoming protocol address or options is not sufficient to store the information. The unit data information to be returned in *unitdata* is discarded.
[TLOOK]	An asynchronous event has occurred on the transport endpoint and requires immediate attention.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

fcntl(2), t_alloc(3xti), t_open(3xti), t_rcvuderr(3xti), t_sndudata(3xti)

Name

t_rcvuderr – receive a unit error indication

Syntax

#include <xti.h>

int t_rcvuderr(*fd, uderr*)
int *fd*;
struct t_uderr **uderr*;

Arguments

fd Identifies the local transport endpoint through which the error report is received.

uderr Points to a **t_uderr** structure containing the following members:

```
struct netbuf addr;
struct netbuf opt;
long error;
```

On return from this call, the members have the following meanings:

addr Specifies the destination protocol address of the erroneous data unit.

opt Identifies protocol-specific options that were associated with the data unit.

error Specifies a protocol-dependent error code.

Description

This function is used in connectionless mode to receive information concerning an error on a previously sent data unit and should be issued following a unit data error indication. It informs the transport user that a data unit with a specific destination address and protocol options produced an error.

The *maxlen* field of *addr* and *opt* must be set before issuing this function to indicate the maximum size of the buffer for each.

If the user does not care to identify the data unit that produced an error, *uderr* may be set to NULL, and `t_rcvuderr()` simply clears the error indication without reporting any information to the user.

Parameters	Before Call	After Call
fd	x	/
uderr->addr.maxlen	x	/
uderr->addr.len	/	x
uderr->addr.buf	x	(x)
uderr->opt.maxlen	x	/
uderr->opt.len	/	x
uderr->opt.buf	x	(x)
uderr->error	/	x

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[BADF]	The specified file descriptor does not refer to a transport endpoint.
[TNOUDERR]	No unit data error indication currently exists on the specified transport endpoint.
[TBUFOVFLW]	The number of bytes allocated for the incoming protocol address or options is not sufficient to store the information. The unit data error information to be returned in *uderr* will be discarded.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

t_rcvudata(3xti), t_sndudata(3xti)

Name

t_snd – send data or expedited data over a connection

Syntax

```
#include <xti.h>

int t_snd(fd, buf, nbytes, flags)
int fd;
char *buf;
unsigned nbytes;
int flags;
```

Arguments

fd Identifies the local transport endpoint over which data should be sent.

buf Points to the user data.

nbytes Specifies the number of bytes of user data to be sent.

flags Specifies any optional flags described below:

T_EXPEDITED
If set in *flags*, the data is sent as expedited data and is subject to the interpretations of the transport provider.

T_MORE If set in *flags*, this indicates to the transport provider that the transport service data unit (TSDU) or expedited transport service data unit (ETSDU) is being sent through multiple `t_snd()` calls. Each `t_snd()` with the T_MORE flag set indicates that another `t_snd()` follows with more data for the current TSDU. The end of TSDU or ETSDU is identified by a `t_snd()` call with the T_MORE flag not set. Use of T_MORE enables a user to break up large logical data units without losing boundaries of those units at the other end of the connection. The flag implies nothing about how the data is packaged for transfer below the transport interface. If the transport provider does not support the concept of a TSDU as indicated in the *info* argument on return from `t_open()` or `t_getinfo()`, the T_MORE flag is not meaningful and should be ignored.

Description

This function is used to send either normal or expedited data.

By default, `t_snd()` operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if O_NONBLOCK is set by means of `t_open()` or `fcntl()`, `t_snd()` executes in asynchronous mode, and fails immediately, if there

are flow control restrictions. The process can arrange to be informed when the flow control restrictions are cleared by means of `t_look()`.

On successful completion, `t_snd()` returns the number of bytes accepted by the transport provider. Normally, this equals the number of bytes specified in *nbytes*. However, if O_NONBLOCK is set, it is possible that only part of the data is accepted by the transport provider. In this case, `t_snd()` returns a value that is less than the value of *nbytes*. If *nbytes* is zero and sending of zero octets is not supported by the underlying transport service, the `t_snd()` returns –1 with **t_errno** set to [TBADDATA].

The size of each TSDU or ETSDU must not exceed the limits of the transport provider as returned in the TSDU or ETSDU fields of the *info* argument of `t_open()` or `t_getinfo()`. Failure to comply results in protocol error (see [TSYSERR] under the DIAGNOSTICS section).

The error [TLOOK] may be returned to inform the process that an event, such as a **disconnect**, has occurred.

It is important to remember that the transport provider treats all users of a transport endpoint as a single user. Therefore if several processes issue concurrent `t_snd()` calls, then the different data may be intermixed.

Parameters	Before Call	After Call
fd	x	/
buf	x(x)	/
nbytes	x	/
flags	x	/

Return Value

Upon successful completion, **t_errno** returns the number of bytes accepted by the transport provider. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

In asynchronous mode, if the number of bytes accepted by the transport provider is less than the number of bytes requested, this may indicate that the transport provider is blocked due to flow control.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TOUTSTATE]	The function was issued in the wrong sequence on the transport endpoint referenced by *fd*.
[TBADFLAG]	An invalid flag was specified.
[TFLOW]	O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting any data at this time.
[TBADDATA]	Illegal amount of data: zero octets is not supported.

[TLOOK]	An asynchronous event has occurred on the transport endpoint.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function. A protocol error may not cause **t_errno** to fail until a subsequent access of the transport endpoint.

See Also

`t_getinfo`(3xti), `t_open`(3xti), `t_rcv`(3xti)

t_snddis(3xti)

Name

t_snddis – send user-initiated disconnect request

Syntax

```
#include <xti.h>

int t_snddis(fd, call)
int fd;
struct t_call*call;
```

Arguments

fd Identifies the local transport endpoint of the connection.

call Specifies information associated with the abortive release.

Call points to a **t_call** structure which contains the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
```

Description

This function is used to initiate an abortive release on an already established connection or to reject a connect request.

Parameters	Before Call	After Call
fd	x	/
call->addr.maxlen	x	/
call->addr.len	x	/
call->addr.buf	/	/
call->opt.maxlen	/	/
call->opt.len	/	/
call->opt.buf	/	/
call->udata.maxlen	/	/
call->udata.len	x	/
call->udata.buf	?(?)	/
call->sequence	?	/

The values in *call* have different semantics, depending on the context of the call to `t_snddis()`. When rejecting a connect request, *call* must be non-NULL and contain a valid value of *sequence* to uniquely identify the rejected connect indication to the transport provider. The *sequence* parameter is only meaningful, if the transport connection is in the T_INCON state. The *addr* and *opt* fields of *call* are ignored. In all other cases, *call* needs be used only when data is being sent with the disconnect request. The *addr*, *opt*, and *sequence* fields of the `t_call()` structure are ignored. If the user does not wish to send data to the remote user, the value of *call* can be NULL.

The *udata* field specifies the user data to be sent to the remote user. The amount of user data must not exceed the limits supported by the transport provider as returned in the *discon* field of the *info* argument of `t_open()` or `t_getinfo()`. If the *len* field of the *udata* is zero, no data is sent to the remote user.

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TOUTSTATE]	The function was issued in the wrong sequence on the transport endpoint referenced by *fd*.
[TBADDATA]	The amount of user data specified was not within the bounds allowed by the transport provider. Some outbound data queued for this endpoint can be lost.
[TBADSEQ]	An invalid sequence number was specified, or a NULL call structure was specified when rejecting a connect request. Some outbound data queued for this endpoint can be lost.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

t_connect(3xti), t_getinfo(3xti), t_listen(3xti), t_open(3xti)

t_sndrel (3xti)

Name

t_sndrel – initiate an orderly release

Syntax

#include <xti.h>

int t_sndrel(*fd***)**
int *fd*;

Arguments

fd Identifies the local transport endpoint where the connection exists.

Description

This function is used to initiate an orderly release of a transport connection and indicates to the transport provider that the transport user has no more data to send. After issuing `t_sndrel()`, the user can not send any more data over the connection. However, a user can continue to receive data if an orderly indication has not been received.

This function is an optional service of the transport provider and is only supported if the transport provider returned service type T_COTS_ORD on `t_open()` or `t_getinfo()`.

Parameters	Before Call	After Call
fd	x	/

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF] The specified file descriptor does not refer to a transport endpoint.

[TOUTSTATE] The function was issued in the wrong sequence on the transport endpoint referenced by *fd*.

[TFLOW] O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting the function at this time.

[TLOOK] An asynchronous event has occurred on the transport endpoint referenced by *fd* and requires immediate attention.

[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

t_getinfo(3xti), t_open(3xti), t_rcvrel(3xti)

t_sndudata (3xti)

Name

t_sndudata – send a data unit

Syntax

```
#include <xti.h>

int t_sndudata(fd, unitdata)
int fd;
struct t_unitdata *unitdata;
```

Arguments

fd Identifies the local transport endpoint through which data will be sent.

unitdata Points to a **t_unitdata** structure containing the following members:

```
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
```

The members have the following meanings:

addr Specifies the protocol address of the destination user.

opt Identifies protocol-specific options that the user wants associated with the request.

udata Specifies the user data to be sent.

Description

This function is used in connectionless mode to send a data unit to another transport user.

Parameters	Before Call	After Call
fd	x	/
unitdata->addr.maxlen	/	/
unitdata->addr.len	x	/
unitdata->opt.maxlen	/	/
unitdata->opt.len	x	/
unitdata->opt.buf	?(?)	/
unitdata->udata.maxlen	/	/
unitdata->udata.len	x	/
unitdata->udata.buf	x(x)	/

If the *len* field of *udata* is zero, and sending of zero octets is not supported by the underlying transport service, the `t_sndudata()` returns –1 with **t_errno** set to [TBADDATA].

By default, `t_sndudata()` operates in synchronous mode and may wait if flow control restrictions prevent the data from being accepted by the local transport provider at the time the call is made. However, if O_NONBLOCK is set by means of `t_open()` or `fcntl()`, `t_sndudata()` executes in asynchronous mode and

fails under such conditions. The process can arrange to be notified of the clearance of a flow control restriction by means of `t_look()`.

If the amount of data specified in *udata* exceeds the TSDU size as returned in the *tsdu* field of the *info* argument of `t_open()` or `t_getinfo()`, the provider generates a protocol error. See [TSYSERR] under the DIAGNOSTICS section. If `t_sndudata()` is issued before the destination user has activated its transport endpoint, the data unit can be discarded.

Return Value

Upon successful completion, a value of 0 is returned. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TOUTSTATE]	The function was issued in the wrong sequence on the transport endpoint referenced by *fd*.
[TFLOW]	O_NONBLOCK was set, but the flow control mechanism prevented the transport provider from accepting any data at this time.
[TBADDATA]	Illegal amount of data; zero octets are not supported.
[TLOOK]	An asynchronous event has occurred on the transport endpoint.
[TNOTSUPPORT]	This function is not supported by the underlying transport provider.
[TSYSERR]	A system error has occurred during execution of this function. A protocol error cannot cause `t_sndudata()` to fail until a subsequent access of the transport endpoint.

See Also

fcntl(2), t_alloc(3xti), t_open(3xti), t_rcvudata(3xti), t_rcvuderr(3xti)

t_sync (3xti)

Name

t_sync – synchronize transport library

Syntax

#include <xti.h>

int t_sync(*fd***)**
int *fd***;**

Arguments

fd Identifies the local transport endpoint.

Description

For the transport endpoint specified by *fd*, `t_sync()` synchronizes the data structures managed by the transport library with information from the underlying transport provider. In doing so, `t_sync()` can convert an uninitialized file descriptor to an initialized transport endpoint, by updating and allocating the necessary library data structures. The file descriptor, which is assumed to have referenced a transport endpoint, has to be obtained by means of an `open()`, `dup()`, or be the result of a fork and `exec()`. The function also allows two cooperating processes to synchronize their interaction with a transport provider.

For example, if a process forks a new process and issues an `exec()`, the new process must issue a `t_sync()` to build the private library data structure associated with a transport endpoint and to synchronize the data structure with the relevant provider information.

It is important to remember that the transport provider treats all users of a transport endpoint as a single user. If multiple processes are using the same endpoint, they should coordinate their activates so as not to violate the state of the transport endpoint. The `t_sync()` function returns the current state of the transport endpoint to the user, thereby enabling the user to verify the state before taking further action. This coordination is valid only among cooperating processes; it is possible that a process or an incoming event could change the endpoint's state after a `t_sync()` is issued.

Parameters	Before Call	After Call
fd	x	/

Return Value

Upon successful completion, `t_sync` returns the state of the transport endpoint. On failure, a value of –1 is returned, and **t_errno** is set to indicate the error. The state returned is one of the following:

T_IDLE Idle

T_OUTCON
Outgoing connection pending

T_INCON Incoming connection pending

T_DATAXFER
Data transfer

T_OUTREL
Outgoing orderly release (waiting for an orderly release indication).

T_INREL Incoming orderly release (waiting for an orderly release request)

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint. This error may be returned when the *fd* has been previously closed or an erroneous number may have been passed to the call.
[TSTATECHNG]	The transport endpoint is undergoing a state change.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

dup(2), exec(2), fork(2), open(2)

t_unbind(3xti)

Name

t_unbind – disable a transport endpoint.

Syntax

#include <xti.h>

int t_unbind(*fd***)**
int *fd***;**

Arguments

fd Identifies the transport endpoint that the `t_unbind()` function disables.

Description

The `t_unbind()` function disables the transport endpoint specified by *fd* that was previously bound by `t_bind()`. On completion of this call, no futher data or events destined for this transport endpoint are accepted by the transport provider.

Parameters	Before Call	After Call
fd	x	/

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned, and **t_errno** is set to indicate the error.

Diagnostics

On failure, **t_errno** is set to one of the following:

[TBADF]	The specified file descriptor does not refer to a transport endpoint.
[TOUTSTATE]	The function was issued in the wrong sequence.
[TLOOK]	An asynchronous event has occurred on the transport endpoint.
[TSYSERR]	A system error has occurred during execution of this function.

See Also

t_bind(3xti)